Лабораторная работа 11

Модель системы массового обслуживания M|M|1

Оразгелдиев Язгелди

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы

Реализовать модель M|M|1 в CPNTools

# 2 Задание

* Реализовать модель системы массового обслуживания M|M|1
* Настроить мониторинг параметров моделируемой системы и нарисовать графики очереди

# 3 Выполнение лабораторной работы

В систему поступает поток заявок двух типов, распределённый по пуассоновскому закону. Заявки поступают в очередь сервера на обработку. Дисциплина очереди - FIFO. Если сервер находится в режиме ожидания (нет заявок на сервере), то заявка поступает на обработку сервером.

Будем использовать три отдельных листа: на первом листе опишем граф системы, на втором — генератор заявок , на третьем — сервер обработки заявок . Сеть имеет 2 позиции (очередь — Queue, обслуженные заявки — Complited) и два перехода (генерировать заявку — Arrivals, передать заявку на обработку серверу — Server). Переходы имеют сложную иерархическую структуру, задаваемую на отдельных листах модели (с помощью соответствующего инструмента меню — Hierarchy). Между переходом Arrivals и позицией Queue, а также между позицией Queue и переходом Server установлена дуплексная связь. Между переходом Server и позицией Complited — односторонняя связь.

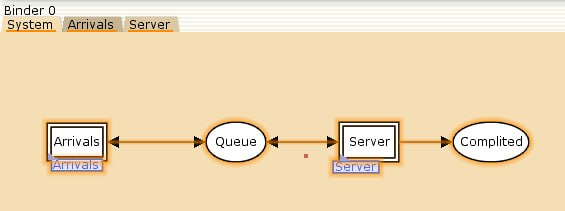


Рис. 1: Граф сети системы обработки заявок в очереди

Граф генератора заявок имеет 3 позиции (текущая заявка — Init, следующая заявка — Next, очередь — Queue из листа System) и 2 перехода (Init — определяет распределение поступления заявок по экспоненциальному закону с интенсивностью 100 заявок в единицу времени, Arrive — определяет поступление заявок в очередь).

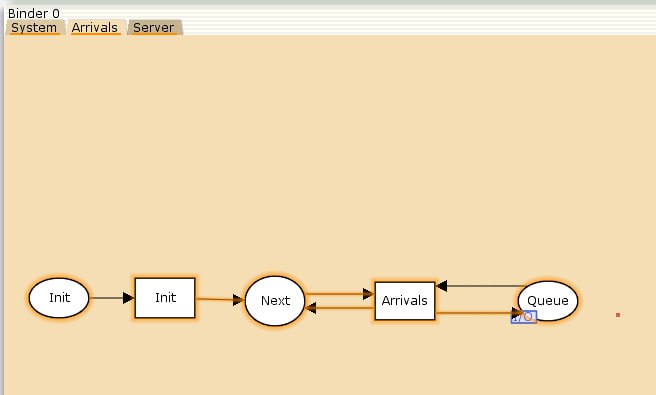


Рис. 2: Граф генератора заявок системы

Граф процесса обработки заявок на сервере имеет 4 позиции (Busy — сервер занят, Idle — сервер в режиме ожидания, Queue и Complited из листа System) и 2 перехода (Start — начать обработку заявки, Stop — закончить обработку заявки)

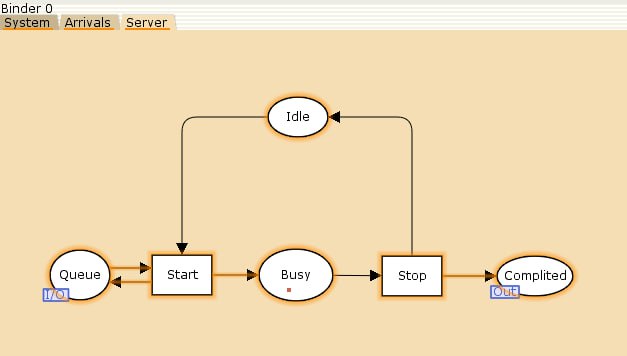


Рис. 3: Граф процесса обработки заявок на сервере системы

Зададим декларации системы. Определим множества цветов системы (colorset): - фишки типа UNIT определяют моменты времени; - фишки типа INT определяют моменты поступления заявок в систему. - фишки типа JobType определяют 2 типа заявок — A и B; - кортеж Job имеет 2 поля: jobType определяет тип работы (соответственно имеет тип JobType, поле AT имеет тип INT и используется для хранения времени нахождения заявки в системе; - фишки Jobs — список заявок; - фишки типа ServerxJob — определяют состояние сервера, занятого обработкой заявок. Переменные модели: - proctime — определяет время обработки заявки; - job — определяет тип заявки; - jobs — определяет поступление заявок в очередь. var proctime : INT; var job: Job; var jobs: Jobs;

Определим функции системы: - функция expTime описывает генерацию целочисленных значений через интервалы времени, распределённые по экспоненциальному закону; - функция intTime преобразует текущее модельное время в целое число; - функция newJob возвращает значение из набора Job — случайный выбор типа заявки (A или B)



Рис. 4: Задание декларации системы

Зададим параметры модели на графах сети. На листе System : - у позиции Queue множество цветов фишек — Jobs; начальная маркировка 1`[] определяет, что изначально очередь пуста. - у позиции Completed множество цветов фишек — Job.

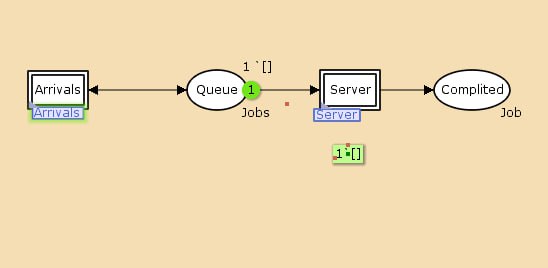


Рис. 5: Параметры элементов основного графа системы обработки заявок в очереди

На листе Arrivals : - у позиции Init: множество цветов фишек — UNIT; начальная маркировка 1`()[**0?**] определяет, что поступление заявок в систему начинается с нулевого момента времени; - у позиции Next: множество цветов фишек — UNIT; - на дуге от позиции Init к переходу Init выражение () задаёт генерацию заявок; - на дуге от переходов Init и Arrive к позиции Next выражение ()@+expTime(100) задаёт экспоненциальное распределение времени между поступлениями заявок; - на дуге от позиции Next к переходу Arrive выражение () задаёт перемещение фишки; - на дуге от перехода Arrive к позиции Queue выражение jobs^^job задает поступление заявки в очередь; - на дуге от позиции Queue к переходу Arrive выражение jobs задаёт обратную связь.

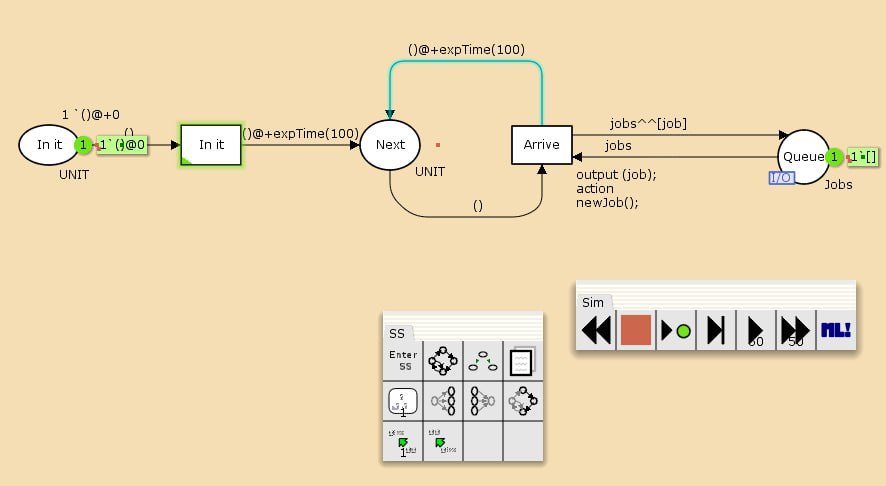


Рис. 6: Параметры элементов генератора заявок системы

На листе Server : - у позиции Busy: множество цветов фишек — Server, начальное значение маркировки — 1`server@0 определяет, что изначально на сервере нет заявок на обслуживание; - у позиции Idle: множество цветов фишек — ServerxJob; - переход Start имеет сегмент кода output (proctime); action expTime(90); определяющий, что время обслуживания заявки распределено по экспоненциальному закону со средним временем обработки в 90 единиц времени; - на дуге от позиции Queue к переходу Start выражение job::jobs определяет, что сервер может начать обработку заявки, если в очереди есть хотя бы одна заявка; - на дуге от перехода Start к позиции Busy выражение (server,job)@+proctime запускает функцию расчёта времени обработки заявки на сервере; - на дуге от позиции Busy к переходу Stop выражение (server,job) говорит о завершении обработки заявки на сервере; - на дуге от перехода Stop к позиции Completed выражение job показывает, что заявка считается обслуженной; - выражение server на дугах от и к позиции Idle определяет изменение состояние сервера (обрабатывает заявки или ожидает); - на дуге от перехода Start к позиции Queue выражение jobs задаёт обратную связь.

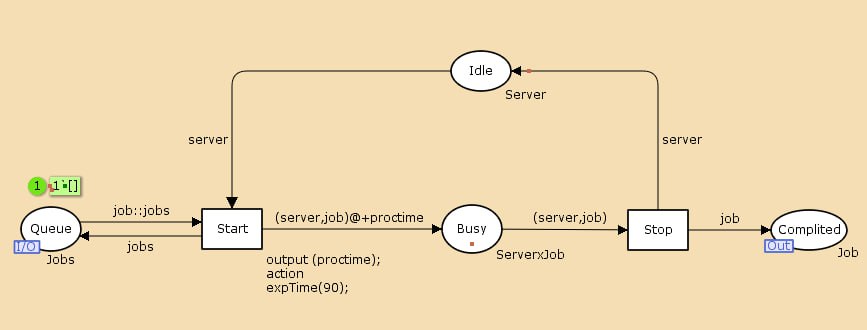


Рис. 7: Параметры элементов обработчика заявок системы

Мониторинг параметров очереди системы M|M|1. Потребуется палитра Monitoring. Выбираем Break Point (точка останова) и устанавливаем её на переход Start. После этого в разделе меню Monitor появится новый подраздел, который назовём Ostanovka. В этом подразделе необходимо внести изменения в функцию Predicate, которая будет выполняться при запуске монитора. Изначально, когда функция начинает работать, она возвращает значение true, в противном случае — false. В теле функции вызывается процедура predBindElem, которую определяем в предварительных декларациях. Зададим число шагов, через которое будем останавливать мониторинг. Для этого true заменим на Queue\_Delay.count()=200

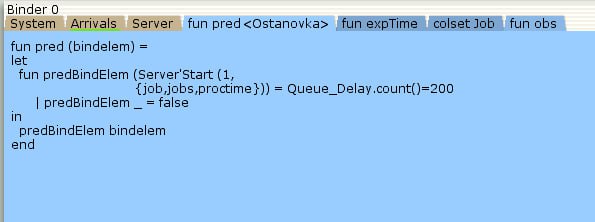


Рис. 8: Функция Predicate монитора Остановка

Необходимо определить конструкцию Queue\_Delay.count(). С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start.Появившийся в меню монитор называем Queue Delay

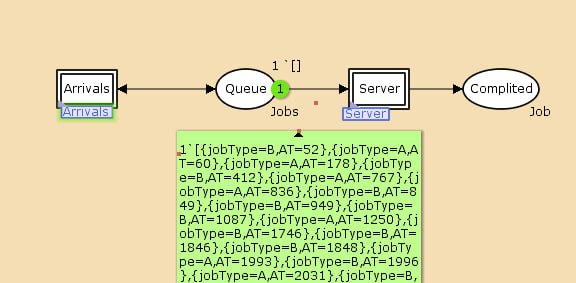


Рис. 9: Запуск системы обработки заявок в очереди

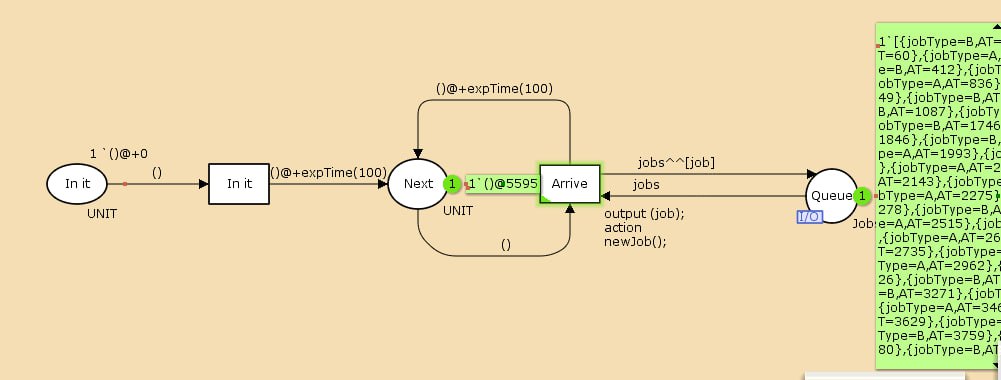


Рис. 10: Запуск системы обработки заявок в очереди

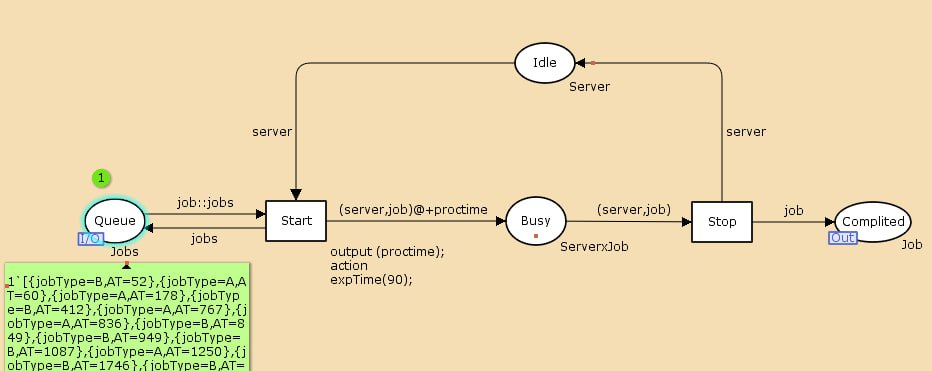


Рис. 11: Запуск системы обработки заявок в очереди

Функция Observer выполняется тогда, когда функция предикатора выдаёт значение true. По умолчанию функция выдаёт 0 или унарный минус (~1), подчёркивание обозначает произвольный аргумент. Изменим её так, чтобы получить значение задержки в очереди. Для этого необходимо из текущего времени intTime() вычесть временную метку AT, означающую приход заявки в очередь

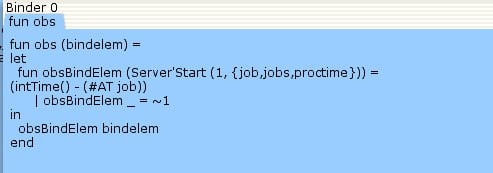


Рис. 12: Функция Observer монитора Queue Delay

После запуска программы на выполнение в каталоге с кодом программы появится файл Queue\_Delay.log, содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время.

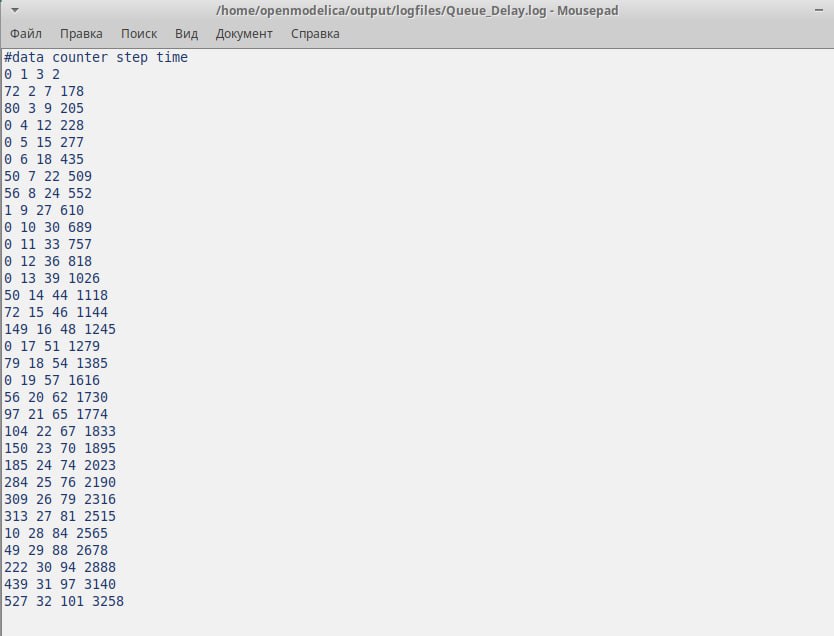


Рис. 13: Файл Queue\_Delay.log

С помощью gnuplot можно построить график значений задержки в очереди (рис. 11.10), выбрав по оси x время, а по оси y — значения задержки:

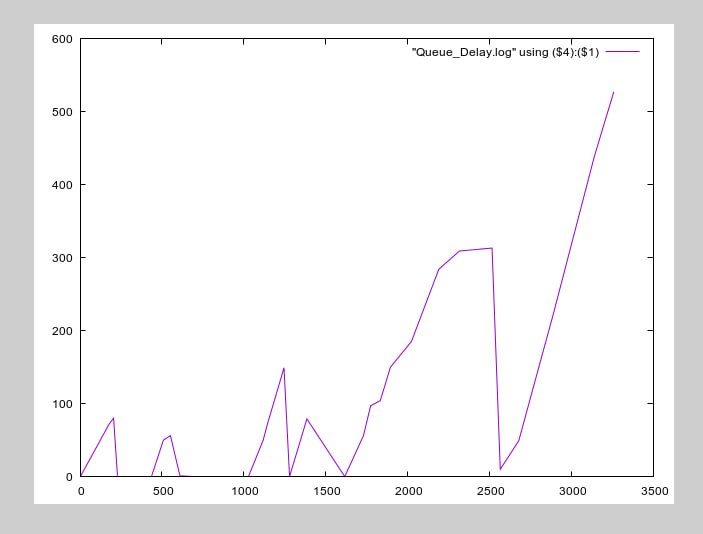


Рис. 14: График изменения задержки в очереди

Посчитаем задержку в действительных значениях. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Появившийся в меню монитор называем Queue Delay Real. Функцию Observer изменим следующим образом

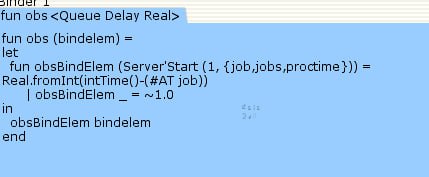


Рис. 15: Функция Observer монитора Queue Delay Real

По сравнению с прошлым описанием функции добавлено преобразование занчения функции из целого в действительное, при этом obsBindElem \_ принимает значение ~1.0. После запуска программы на выполнение в каталоге с кодом программы появится файл Queue\_Delay\_Real.log с содержимым, аналогичным содержимому файла Queue\_Delay.log, но значения задержки имеют действительный тип.

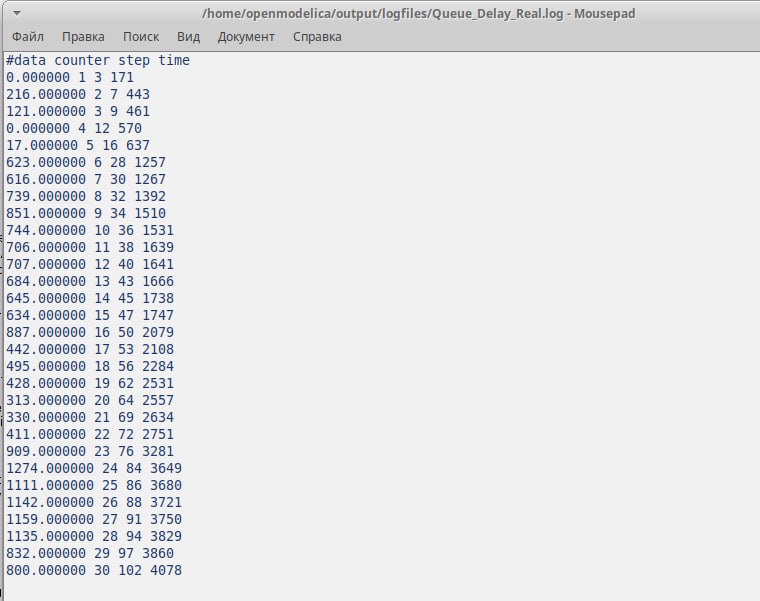


Рис. 16: Файл Queue\_Delay\_Real.log

Посчитаем, сколько раз задержка превысила заданное значение. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Монитор называем Long Delay Time.

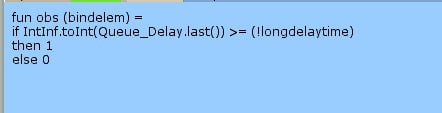


Рис. 17: Функция Observer монитора Long Delay Time

Если значение монитора Queue Delay превысит некоторое заданное значение, то функция выдаст 1, в противном случае — 0. Восклицательный знак означает разыменование ссылки.

При этом необходимо в декларациях (рис. 11.13) задать глобальную переменную (в форме ссылки на число 200)

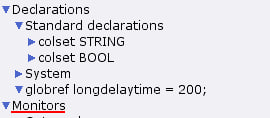


Рис. 18: Функция Observer монитора Long Delay Time

После запуска программы на выполнение в каталоге с кодом программы появится файл Long\_Delay\_Time.log

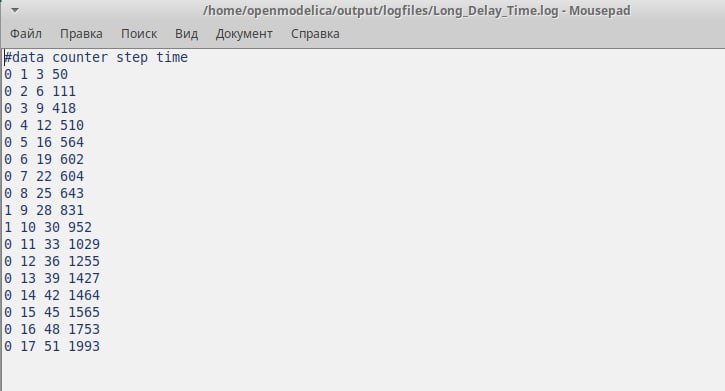


Рис. 19: Файл Long\_Delay\_Time.log

С помощью gnuplot можно построить график , демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200

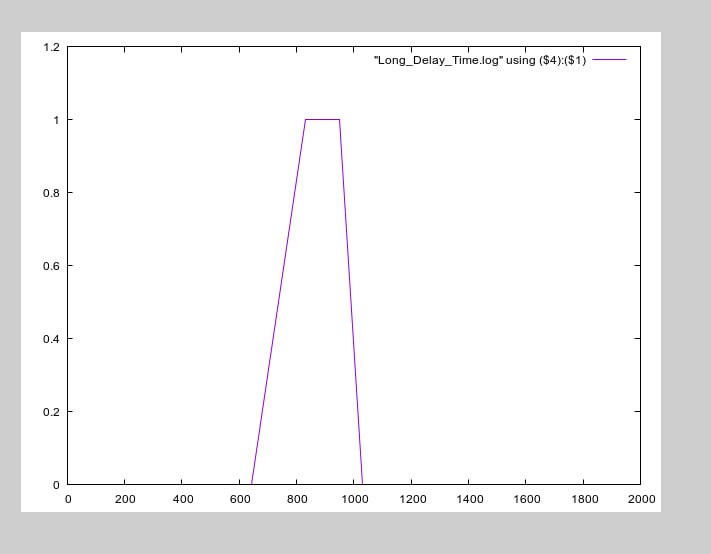


Рис. 20: Периоды времени, когда значения задержки в очереди превышали заданное значение

# 4 Выводы

В процессе лабораторной работы мы реализовали модель системы массового обслуживания M|M|1 в CPNTools